EPISODE 1083

[INTRODUCTION]

[00:00:00] JM: A frontend developer issuing a query to a backend server typically requires that developer to issue that query through an ORM, or a raw database query. Prisma is an alternative to both of these data access patterns allowing for easier data access through auto-generated type-safe query building tailored to an existing database schema.

By integrating with Prisma, the developer gets a database client that has query auto completion and an API server with less boiler plate code. Prisma also has a system called Prisma Migrate, which simplifies database and schema migrations.

Johannes Schickling is CEO of Prisma and he joins the show to talk about the developments of Prisma that have occurred since we last spoke and where the company is headed.

If you want to reach to 30,000 unique engineers every day, consider sponsoring Software Engineering Daily. Whether you are hiring engineers or selling a product to engineers, Software Engineering Daily is a great place to reach talented engineers. You can send me an email, jeff@softwareengineeringdaily if you are curious about sponsoring the podcast. We're also looking for writers and a videographer. If you're great with video or you're a great writer and you want to write about software, then send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

[00:01:34] JM: Scaling a SQL cluster has historically been a difficult task. CockroachDB makes scaling your relational database much easier. CockroachDB is a distributed SQL database that makes it simple to build resilient, scalable applications quickly. CockroachDB is Postgres compatible, giving the same familiar SQL interface that database developers have used for years.

But unlike older databases, scaling with CockroachDB is handled within the database itself so you don't need to manage shards from your client application. Because the data is distributed, you won't lose data if a machine or data center goes down. CockroachDB is resilient and adaptable to any environment. You can host it on-prem, you can run it in a hybrid cloud and you can even deploy it across multiple clouds.

Some of the world's largest banks and massive online retailers and popular gaming platforms and developers from companies of all sizes trust CockroachDB with their most critical data. Sign up for a free 30-day trial and get a free T-shirt at cockroachlabs.com/sedaily.

Thanks to Cockroach Labs for being a sponsor, and nice work with CockroachDB.

[INTERVIEW]

[00:02:57] JM: Johannes, welcome back to the show.

[00:02:59] JS: Hey, Jeff. Great to be back on the show. Thanks so much for having me.

[00:03:02] JM: Of course. You run Prisma, and Prisma is involved in workflows for accessing data. Can you describe the APIs that sit between the frontend and the backend database layer and where Prisma fits in?

[00:03:20] JS: Sure. I think that's a pretty complex questions and it always depends on what your application architecture looks like. There are so many angles to take this if you'd, for example, take a more modern Jamstack architecture or if you take a microservices architecture. The answer here is always it depends. What's always the same is if you build an application that requires data persistence, then chances are you're using a database.

How Prisma fits into that is that it tries to help application developers build applications more easily working with the databases. Typically, that means that you're using a part of Prisma what's called the Prisma client that sits typically in your application server. That's typically an API server and talks to your database.

Typically, this part of the stack is known as an ORM layer, or a data access layer. Prisma in particular is not an ORM. We can talk about that separately. That's a pretty nuanced topic, but

Prisma typically, the main function is to serve to access data noisily in your application language.

[00:04:28] JM: Can you talk about that in more detail? Why would I need an additional layer of access? I mean, I think, in general, if I'm sitting on the frontend and I want to access the database, I am hitting some service. That service is talking to a database and the service is requesting the data from the database. Why do I need Prisma to help out with that database access?

[00:04:57] JS: In this setup, just to recap one more time. You have your frontend application. Let's say you have like React, a Vue app. On the other end, you have a database. Let's say you have a more traditional Postgres, MySQL database but would also apply the same for more modern DynamoDB, etc. And then typically you have this middle tier that's, let's say, an API server. Where you would use Prisma for is just having an easier time building your API server in order to talk to the database.

Let's say you're Postgres. The most bare-bone thing you could do is implementing your API server and just writing implementing your endpoints, or let's say you're building a GraphQL server, implementing your resolvers and then just talking directly to the database by writing raw SQL queries. That works, but that comes also with some problems typically in terms of productivity and like there's not quite the abstraction level that you want as an application developer to be productive and confident in what you're writing.

The same way as frontend applications are built through abstraction layers, let's say React, Angular, Vue. It's the same on the backend that you also want more application idiomatic abstraction layer for the way how you're talking to the database. Historically, there're been many forms of that. The most common one is in ORM, and there are more modern ways of how you build a better abstraction on top of your database for data access, and that's a pattern that we're implementing with Prisma that typically you refer to as a query builder.

[00:06:32] JM: Can you explain in more detail what is the difference between a query builder and an ORM?

[00:06:37] JS: Right. That comes down to the way how you're thinking about these application patterns. An ORM stands for object relational mapper, and the idea behind an ORM is mapping typically a database table to a class in typically object-oriented programming language. This is a pretty intuitive model and is widely used in tons of ORMs. The most prominent one probably being active record as part of Ruby on Rails, but there are tons of other ones as well. In the Java world, there's Hibernate, and the idea there is you have tons of tables in your database and you want to map that somehow into your programming language. In your programming language, you are typically working with classes.

As supposed to a query builder, which looks a lot more like SQL in a certain way, but maps these SQL statements into statements in your programing language. The differences really come down to how much flexibility and control you needs, but there are tons of downsides of ORMs, that as ORMs got more widely used, became more and more well-known.

There is a great blog post called the Vietnam of Computer Science, which is all about ORMs and the problems behind ORMs. Most importantly, there's one thing called the object relational impedance mismatch, which talks about the problems of mapping databases, database tables to objects where there is just a big amount of an impedance mismatch.

The way around that is that you should rather think about the queries that you're writing to a database instead of obsessing too much about the classes and the objects and your query should really determine the shape of the data you're getting back in the same way as – The pretty striking analogy to how GraphQL deals with this sort of pattern were GraphQL was all about the query that you're writing that you need in your components and it's a pretty similar pattern that you are now applying the way how you do data access on the backend.

[00:08:51] JM: If I was to setup Prisma for my application, what would the life of a query look like and the structure of a query look like?

[00:09:08] JS: What you'd be using concretely there is Prisma overall is a database toolkit, and what you would use to query a database is a part of Prisma called the Prisma client. The Prisma client is basically just a JavaScript library that you install from npm and you're basically writing that query once in your code. One great advantage is that it's fully type safe by leveraging typescript. So you’re writing that query, and then throughout runtimes, when your application is deployed, that code gets invokes. That under the hood generates a database dependent query, typically a SQL query, but as we’re supporting of the database as well, it could generate Dynamo queries, etc. These queries are then sent to the underlying database and the data is returned and then returned into your application code.

[00:10:02] JM: Got it. What's the difference between using Prisma and using GraphQL?

[00:10:10] JS: It is a really two fundamentally different technologies for different use cases. I think a good way to think about it is where in the application stack these technologies sit. GraphQL is typically used for frontend applications to talk to backend applications. Whereas Prisma, and specifically the Prisma client, is used for typically your backend application to talk to your database. They’re analogous in this way, but typically sit at different layers of the stack.

However, it always depends with newer approaches like the Jamstack. Your frontend application can statically directly talk to your database making the API server layer obsolete and you could use Prisma directly for that. But the most common use case is more of like three-tier architecture where you would use Prisma to talk from your application API server to your database.

[00:11:01] JM: Got it. What's the benefit of – Or like the companies that you work with that have put Prisma into their backend and they use that as a way to have the backend database access function, what improvements does that lead to for their architecture?

[00:11:23] JS: I’d like to reframe that conversation a little bit and in order to talk more about like which problem does it solve. The problem it solves is – The question here is very related to what problems does a better abstraction layer solve? I think it solves a couple of problems. One is a problem of productivity that you could do something in a certain way, but there is a better way to do it because you're more productive building your applications and you have more confidence in how you’re building it.

Prisma provides a better abstraction layer on top of your databases for any kind of workflow, and for data axis in particular, let's say you’re building a modern backend application. And where we see a lot of adaption for is particularly NodeJS and particularly in Typescript. Typescript just has so much adaption right now, and typescript is all about type safety. It’s really, really difficult to bring the schema layer from your database into the type safe environment of your programming language.

In this case, in particular, Prisma enables by letting Prisma introspect your database schema and generate kind of like a bespoke custom-tailored database client library that slots right into your application framework. That would be one big one.

Another one is working with databases is pretty difficult. It's one thing that you write the right query to get the data back that you need, but then there's also 20 ways to do that. Probably just a few of these ways is actually performant. Making database access performant is another big, big problem that we can optimize in many, many ways. Helping you with problems called the N+1 query problem, for example, or if you're building, let's say you implement a GraphQL server. There's tons of optimizations that you need to do, and all of these things we optimize by having build a custom-tailored query engine that sits on top of a database.

Another one, which we don’t yet support, but we are planning to roll out support for a little bit further down the road is in order to support accessing multiple databases at the same time. We see especially in larger companies polyglot persistence being more and more common in application architectures. That your data is fragmented between, let’s say, Postgres and Elastic, but your application still wants a more unified way to accessing that data. That's another use case for Prisma further down the road.

[00:14:04] JM: In these cases, if I am issuing a query on Prisma in my backend, where is that Prisma query getting interpreted into the native database query?

[00:14:21] JS: Right. Again, taking a little step back, you would use Prisma not just for data access, but you could use it for many different workflows, so also to migrate your database schema, etc. But in that particular case of accessing your database, Prisma supports multiple different database connectors. Each of them are implemented separately through a plugin architecture.

The way how these underlying database queries are generated depends on which database connector you're using. Right now we are supporting Postgres, MySQL, SQLite. We're working on MongoDB. We're working on mSQL, and over time we want to cover support for any kind of database and any kind of data source really where you could think of notion, for example, being a data source. Once a query gets executed on a Prisma client level, it invokes the Prisma query engine under the hood. That right now is running as a separate process, but over time we'll be hosting that directly within the programming language WebAssembly. That query engine has the responsibility of translating this Prisma client query that's typically expressed through a construct in your programming language into the native database queries. That typically would be a SQL query in the most common cases.

What's interesting here is that it doesn't just translate one big Prisma query into one big SQL query, but sometimes it's actually more efficient to break up that one bit query into multiple smaller SQL queries, which is more efficient and scales better. These are patterns that we've seen implemented for companies or web scale companies that need to scale out their databases. These optimizations that we can bake right into the Prisma client'. Application developers just have to write the query in a very intuitive and language-idiomatic DSL, and under the hood we make sure that it gets translated into the most optimized database queries, mostly SQL queries.

[SPONSOR MESSAGE]

[00:16:39] JM: When you spend your spare time learning, you can accelerate your career. O'Reilly lets you learn through high-quality books, videos, courses and interactive experiences. O'Reilly content has been built over decades. They're a trusted source of effective technology education. If you're an individual leveling up on your own, you can use O'Reilly to chart a course for your career goals. If you manage a team or a company, you can get access to O'Reilly's career development resources for your whole organization.

Go to softwareengineeringdaily.com/oreilly to explore O'Reilly's e-learning experiences. You can build the skills you need to future-proof your career. Check out softwareengineeringdaily.com/ oreilly, and thank you to O'Reilly for being a partner with Software Engineering Daily for many years now.

[INTERVIEW CONTINUED]

[00:17:39] JM: The process of setting up my Prisma schema, like if I want to use Prisma for my application, what's that set up process like? How am I defining that schema translation layer?

[00:18:00] JS: I think we can distinguish here in two broader use cases. One would be a complete greenfield use case. Let's say you want to build a completely new app. You're starting from scratch. You're taking all of the pieces of your application and you, for example, decide on Postgres and you want to use Prisma with Postgres and you want to build, let’s say, a REST API node. What you would do is like you would set up your node project. You would install Prisma as an npm dependency and you would then run the Prisma CLI Eli Prisma in it. This would set up a file for you. That's the Prisma schema. In that Prisma schema, you say you want to use Postgres. You configure your database credentials and then you start writing in that Prisma schema your database schema. There're multiple keywords, for example, keyword model where you can say, “I want a model user. I want a model post,” and you express your database schema and then you use the Prisma CLI in order to migrate your database schema. Make sure that all the right tables are created and then also the right Prisma client that just fits your database schema is generated and you can start making database queries. That the greenfield use case and that’s super straightforward and it gives you the great benefit of you know just the right SQL commands in order to create the right tables, but you can declaratively express your database schema similar to a nice analogy here, is like Prisma is kind of like Terraform, but for your database schema. It you can fully declaratively express your database schema. That's the greenfield use case.

Another very common use case is that you want to may be re-factor some of the applications or we see it a lot that people have built applications in Rails and they want to move them to JavaScript. They step-by-step face all their Rails backend, but they want to keep the database. This is where they can use Prisma to introspect their existing database schema. This is one of the workflows that Prisma supports so that instead of you writing your database schema manually, you just tell Prisma, “Hey, these are my database credentials,” and we introspect your database schema, generate the Prisma schema from it, and then generate the Prisma client that matches your existing database schema.

Then you can either at some point say, “No. I actually want Prisma to also take the responsibility of taking care of my database, my schema migrations going forward,” or you say, “I want to not change my database schema, or maybe Rails keeps changing it,” and then you integrate the Prisma introspect workflow into your CICD system. That also gives you that full type safety.

[00:20:52] JM: The actual implementation of Prisma is something I'm curious about. When a Prisma queries getting executed, I know the first version of Prisma, you had a proxy server that sat on top of a database and handled the incoming queries and Prisma 2.0 no longer has that Prisma server. Can you explain the architectural changes that you made to Prisma and the benefits to your change to Prisma 2.0?

[00:21:26] JS: Yeah. What I've been describing so far is all about Prisma 2. Prisma 1, the predecessor of what's available now, has been more bit of an artifacts of our historic origins, which actually started out more on a different architectural paradigm of the backend service, and since then was Prisma. With Prisma 2, we’ve been really now focusing on the database toolkit kind of framework approach. This is where Prisma 1 has still been more like an artifact. That application server and the separate Prisma server has been more of an artifact of the past, and Prisma 2, we’ve build completely from scratch with the best architecture in mind. The Prisma query engine and the Prisma Migration engine are all implemented in Rust, which gives us great performance benefits and lets us integrates the Prisma query engine, Prisma Migration engine seamlessly into various programming languages.

Right now, we support JavaScript and Typescript, but we are already experimentally supporting Go as well and we’ll be covering other languages as well. That's really a design goal for us that we write the tricky part once when comes to the performance optimizations, etc. and that's all in Rust and that’s super battle-tested. Then we can just seamlessly integrate that into these client languages, whether that’s typescript, JavaScript, whether that’s Python further down the road, and that these are basically just thin layers around that that are all generated.

Right now, the way how you deploy the Prisma client is just as a JavaScript library that's generated. There is no intermediate server that you need to deploy separately or you need to manage separately. The way how you run migrations is just throw CLI so you can embed it into your application architecture however you want.

[00:23:28] JM: What are the engineering challenges that you've encountered in the rewrite of Prisma 2.0 when you – You rewrote it to Rust, right?

[00:23:36] JS: That's right.

[00:23:37] JM: Yeah. What's been challenging about that?

[00:23:39] JS: Obviously, the engineering teams would probably be the better people to answer this question, but from what I can tell is that Rust is a fairly complex language. We've been using Scala before and we've been moving it to Rust. Memory management and Rust is notoriously hard. This was quite a learning curve for everyone involved.

Yeah, that was just challenging, but it's been really worth it and we have an immense test coverage. Moving that over has also just like taking a bunch of time. That's sort of on the Rust layer. But the entire point of Prisma is that we make it as language-idiomatic as possible, so also covering all the features that are available in Typescript to give you the best possible experience working in your application language. We've been really pushing the limits there. Have been using Typescript features, interacting with the Typescript team. Sometimes they've been even surprised of like what we enabled and what we made possible. Hit a lot of bugs there, and generally like we want to go through all of the mess in the JavaScript ecosystem in particular that our users don't have to. That means like for every possible edge case you could be hitting there, we've already hit it and implemented some smooth path there, but that also means that we had like every possible combination of regression bugs in node or yar in different version managers. We basically have to – This is an incredible broad spectrum of where JavaScript fatigue, that's real, and we basically deal with the JavaScript fatigue and all the edge cases there that our users don't have to, and ideally just smooth sailing for them. There's maybe not just this one hard challenge, but 10,000 annoying small challenges that we need to have like proper testing for, etc., and that just takes a little while.

[00:25:38] JM: I'd like to understand more about database migrations and the motivation for Prisma Migrate, which is something that allows you to migrate databases. What are the potential scenarios that would require using Prisma Migrate?

[00:25:55] JS: We touched before on the two kind of main use cases that a user might have, either the greenfield application, the greenfield use case or the a brownfield use case. You would use Prisma Migrate particularly in the greenfield use case where you don't have a running database set up before where it’s not in the right state and you want to get it structurally in the right state.

This is the case for both schema-full and schema-less databases. Even if a database says it’s schema-less, then your application basically becomes the schema, and you need to deal with schema migrations in your application. Whenever you change something about the way how you’d store data, there’s still structure to that. Then your application server basically becomes a big pile of if/else statements and becomes a mess, and everyone wants to have that schema basically for your application data.

At the core of these migrations, it’s all the schema that expresses the structure. Then migrations is you’re basically changing your mind of like what that structure should be, and then applying that is incredibly hard, especially incredibly hard if you want to do so with confidence and in a deterministic way.

There are a lot of – That’s a really hard challenge where you want to get every little piece of technology that can help you hold your hands and make sure that you go through that. We’ve heard a lot of customers perform what they call data surgery when they’ve messed up a database migration. It’s a pretty common on a Friday afternoon. Then they basically just – They wish for better tools, and have lots and lots of companies have invested into their own database migration tooling as there’re no good integrated tools available.

This is what we’re working on with Prisma Migrate, where we take that declarative approach. The way how you design your database schema and we derive and generate the imperative migration steps from that that you can check into CICD and apply deterministically to your applications and development to your applications in production.

[00:28:14] JM: Just to ask naïvely, why do database migrations happened? Who needs to migrate the database from one place to another?

[00:28:29] JS: Right. It's a great question, and I think whoever has built an application has experienced that you've – You don't know everything upfront. There is like this other feature that you want to build. For that, you realize, "Oh! I don't have the data for that. I need to store this additional data."

Let's say you built a super simple blogging application and you have a title and you have a body for each blog post, but then you also want to introduce a sub-headline. Then you need to introduce [inaudible 00:29:02] to your database and you need to maybe backfill these for the old items. With your changing business requirements, you need to store that data somewhere, and this is what gets translated. The vehicle how you get that out is through database migrations.

[00:29:21] JM: For what kinds of workloads is the Prisma Migration most well-suited to? What kinds of database migration workloads?

[00:29:33] JS: Database migrations, maybe it's worth also distinguishing here. Database migrations is a pretty overloaded term. What we're talking about here is really migrating your existing database schema. We are not talking about migrating data from an Oracle database in a Cockroach database. We're really talking about someone building an application and their requirements change and they want to change the database schema.

In terms of workloads, it's incredibly wide spectrum, whether you just want to add a little field or whether you add a huge new sub-category of your application. You like hundreds of tables. It's completely flexible. It's a pretty similar model as for the Prisma client data access part where Prisma translates a Prisma query into underlying SQL queries. It's the same here for migrations that you express the desired state you want to be in through the Prisma schema. Then the Prisma Migration engine translates that into the undulating SQL migration steps or other migration steps that database gets in the right shape.

The mental model here is the same way as React has the virtual DOM and you can declaratively describe which stage your React app should be in and then React applies all the divs. This is kind of the same approach here for Prisma or similar to Terraform.

[00:31:07] JM: I'd like to know more about how large companies are using Prisma and why it's been useful to them. How it fits into their architecture? Do you have any case study examples?

[00:31:23] JS: Sure. I think it's pretty analogous to which kinds of applications are built with databases. Databases are used at huge, huge companies and enterprises, but they are also used for your little pet project that you build with Rails in a weekend. We see the entire – The same spectrum of usage within Prisma. Whether it's hobbyists building their new side project and they have no time and they want to get more stuff done faster, and so this is where they want to get as much help as they can get and choose Prisma in order to not waste time on their database, or whether it's a large enterprise that wants to maybe standardize the way how they built backend applications and how they work with their database, or who want to currently go through digital transformation or want to modernize their backend architecture and want to build new applications on existing databases. We really see adoptions throughout the entire spectrum of use cases.

[00:32:31] JM: Do you have like any particular examples, like any deeper examples for what's a particular company that's found a lot of value in it?

[00:32:40] JS: Sure. I think what I can tell you is companies that have been using Prisma 1. We're just in the transition to Prisma 2. Prisma 2 is now been out in beta for a couple of months. Maybe by the time this show airs, Prisma 2 is already available in GA. We see a lot of adaption for the Prisma 2 beta already and even for larger companies using it in production. I'm not sure whether I can tell the names of these companies. What I can tell you more about is companies who've been using Prisma 1 production, which addresses the same use cases.

This is where we had companies like Adidas or Splunk, like couple larger companies really using Prisma in production, and we see just a lot more appetite and a lot more adaption already trending up for Prisma 2. I'm pretty sure we'll be able to tell you a couple of really big names in the coming months, and we're also planning a user conference a little bit further down the road where we'll also want to showcase more.

[00:33:42] JM: I assume this is a virtual user conference.

[00:33:44] JS: That's right.

[00:33:46] JM: It's worth asking you, you're now an entirely remote company. What has been the transition like for you? The process of going from a company with an actual office to becoming fully remote?

[00:34:00] JS: Yeah, that's a really great topic as well. We started out with the company roughly 4 years ago all in Berlin. We had a pretty small cozy office where we're just been a handful of people back then. And over time, we grew. We're right now just suppressed 30 people, and at some point we also realized, "Okay, we will serve a global customer audience and we need to be close to our customers, but we also need to hired the talent, where the best talent is located." In some cases, the best talent has been in Canada, or France, or the US, or elsewhere. We started out with a couple of separate functions. We thought initially are more self-contained, let's say, and started testing the waters there to expanding beyond just a single office in Berlin to also supporting remote.

As then we also went more into executive hiring as well, we also realized, "Okay, we need to hire the best possible people, the best possible people might live somewhere in Kentucky. They might live somewhere in the UK. They might live all around the world." This is where we went beyond just the more self-contained functions to saying this is now what we're doing as a company and this is what we've been basically starting this journey roughly 9 months ago and have, before Corona actually came a thing, I think been on the way to becoming not just a remote-friendly, but a remote-first company. Then this is already the trajectory we've been on. Then as the Corona pandemic unfolded more and more, it has forced us on that trajectory even more.

In that regard, it didn't really catch us all that much by surprise in terms of flipping the way how the company operates, but it basically just accelerated that transition and it's been pretty smooth and I think maybe not everyone's favorite setup to work remotely, but I've heard from many employees that they've been more productive since they can remember and they're having a really great time and I think has worked incredibly well for us so far.

[SPONSOR MESSAGE]

[00:36:33] JM: JFrog SwampUp is an online user conference with more than 30 sessions from cross-industry expert, including Google, Microsoft, capital one, Adobe and more. You can get a unique look into the broad DevOps market, not just point solutions. There will be tracks for cloud native DevOps, enterprise DevOps, DevSecOps and digital transformation.

Participate in expert DevOps training classes across DevOps tools, security, CICD and more, and you could join from two time zones, June 23rd and 24th for the Americas, and June 30th and July 1st for EMEA and APac to suit daylight hours across the globe. Go to softwareengineeringdaily.com/swampup to learn more and sign up

JFrog will be donating all conference registration proceeds to COVIT-19 research. Go to softwareengineeringdaily.com/swampup and check out SwampUp.

[INTERVIEW CONTINUED]

[00:37:38] JM: Revisiting the usage of Prisma, we had a show recently with Tom from RedwoodJS, Tom Preston Warner, and the Redwood JS platform, well, I guess you would call it a framework for deploying Jamstack applications. Can you explain how RedwoodJS uses Prisma and how Prisma fits into the burgeoning vision for Jamstack?

[00:38:10] JS: Sure. That's a great topic. Our overall goal at Prisma is to help application developers build better applications, and the way how we help them in particular is by working better with databases, but redwoods and many other application frameworks are as in line with our vision as what we are working on ourselves, and this is where we feel almost like a B2B vendor for these application frameworks giving them great tools that they can build an better framework for their users.

We almost think in terms of our user base as one, our direct users, who architecturally want to use directly Prisma and they want to more assemble each part of their stack, and then there are users and application developers whom I want more everything out of box in one big framework similar to what Ruby on Rails has been, and is what Redwood is working on. For them, we're basic giving them one building block that takes care of all the database workflows. I think it's really good analogy here if you think of Redwood as Ruby on Rails, then Prisma fits in there as like kind of an active record equivalent.

This is where we're helping the Redwood framework with their use cases, but there're also plenty of other frameworks. There is a new JavaScript framework call Blitz that works on top of the NextJS. There other, for example, GraphQL frameworks. We really want to help all of these application frameworks to work better with databases and are giving them Prisma, basically, a building block to build a better framework.

[00:39:55] JM: To come back to the internals, I think type safety is something that's worth focusing on here. Can you explain how type safety is relevant to Prisma and where – Type safety can exist at all different areas of the stack. I guess it's most important when you're – Well, I mean, has a lot of applications. But explain where type safety fits into Prisma and what value it provides.

[00:40:27] JS: I am a huge, huge fan of anything type safety. I was a huge fan of Swift, Rust, Haskel, anything that gives you powerful type system just because it gives you more confidence. It powers tools that make you more productive and it has basically no downsides besides maybe a slight overhead in terms of build speed, but you can sort of neglect that in a modern environment. But it's really hard to provide great type safety for your application development and experience.

Since type safety, you can't really retrofit type safety for all use cases, particular for databases. These tools have to be designed for type safety from the ground-up, which is what we've done with Prisma. We've designed every API in order so that it can be fully type safe. Since there are so many moving pieces, your database is by definition decoupled from your application and it therefore can drift apart. Your database schema might change, but your application might not have yet caught up with that change. So you want to have some tangible way of seeing like, “Hey, [inaudible 00:41:42] work together.” By allowing you to introspect your database schema, derive an artifact that where now the compiler gives you a confidence and tells you, “Hey, that works, or it doesn't work.” That's absolutely huge especially if you're building microservices and you don’t just have that problem once, but you have thousands of microservices. You want to get every little bit of type safety you can get.

This has been really a core, core part of what we’ve been building and it doesn't just give you that confidence, but it also powers like every part of the tooling that developers love whether it’s the best code typescript integration, just top-notch. All of the auto completion, red squiggly lines, etc. All of that is powered through type safety. Even for the funny symmetry almost in here, that there's tons of developers who don't even know that this is type safety or typescript. They just know I want better auto completion. I want these red squiggly lines. But all of that is powered through that type safety and this is where everything is based on for us.

[00:42:52] JM: Tell me more about the problems in backend access that you would like to address with Prisma and I guess with your company in the future. That's a great question, and I think that this makes room for – Is maybe a nice segue for future shows as well since there is – It’s basically unbounded in that regard that there are many use cases for databases and so many problems developers have with databases.

To give you a couple ambitious ones, for example, if you're working with multiple databases and you have some sort of replication system set up where you want to synchronize data from one to another, for example, in order to implement read-write models. That’s a workflow and a use case that we want to facilitate further. I talked about the distinction between database schema migrations and database migrations in terms of moving from one database to another. That's another one we see ourselves addressing in the future.

Then, also, in regards to polyglot systems is something that we are laying out the foundation for. There will be always more than we can handle and we want to see what does a market most urgently need. What do our users most urgently need, and we build everything in a modular way that the ecosystem – There's already an ecosystem growing around Prisma where developers step in and build ecosystem tools for things that we haven't covered yet, which is also really great to see. Yeah. For now, we really want to focus on the data access, the schema migration part and then all the workflows around that.

[00:44:36] JM: What are the business opportunities there? Do you have any vision for what you might be able to structure a business around?

[00:44:46] JS: Right now, we're a venture funded company. We've, two years ago, raised a seed round. There's yet unannounced follow-up around to that, which we're going to announce at some point this year, but now it's been spoiled through this. We are right now fully focus on building the open-source foundation of Prisma. Prisma will always be open source, the foundation, but this gives us a broad adaption base of users who want to buy an additional commercial complementary product. This might be delivered as a club product. It might be delivered as an on-prem product and might be supplemented through services offering as well, and we are very inspired there through Hashi's philosophy where they separate their open source offering, their commercial offering by everything's open source and free that solves technical complexity and everything that solves organizational complexity is commercial.

We see a very good analogy here, and where we see Prisma offering a commercial solution is as what we think of calling an application data platform. As supposed to data platform for data analysts, more of a platform for application developers that gives them all the workflows like running these applications in production what a CICD for databases look like. What does collaboration look like? You want to have a schema registry for all of your microservices that each have their own databases, and this is where we see a lot of these systems being built at larger companies, whether it's Facebook, or twitter, or LinkedIn, but they're not open source. They're so monolithic and like integral to their application architecture, but we want to take these ideas and make them accessible to everyone as a commercial product as well.

[00:46:41] JM: Very cool. Well, do you have any other closing thoughts on the space that you're building in? Subjects that have come up recently that you're thinking about in just the landscape of data access?

[00:46:57] JS: Yeah, I think the landscape is really what gets me so excited on a day-to-day basis. Since I mentioned the term JavaScript fatigue before, this means basically that's a negative side of it. There is constantly something new, but there is so much innovation going on and so many new technologies and so much drive in that ecosystem, whether it's new database vendors, like Fauna, or Cockroach, or whether it's new application frameworks like Redwood or Blitz, and we basically fit right in the center of all of this where we basically want to be similar to how Reacts is just a foundation of an ecosystem, and like if it works with React, then you're like, "Could you adapt this library?" We're seeing Prisma on a similar trajectory where Prisma just becomes the default way how developers think of working with a database, and this enables this amazing ecosystem that we see flourishing.

Whether it's like the show you've done with Tom Preston Warner and Redwood, or other frameworks in the future. You've done one on NextJS, which we're also integrating deeply with. This is for me the most exciting thing. All the pieces fit together like Lego bricks.

[00:48:10] JM: Johannes, thank you for coming on the show. It's been great talking.

[00:48:12] JS: Thanks so much, Jeff.

[END OF INTERVIEW]

[00:48:22] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[END]